Attorney Docket: Framework/Patent
February 14, 2001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application for United States Patent:

AUTOMATIC TRANSFER AND EXPANSION OF APPLICATION-SPECIFIC DATA FOR DISPLAY AT A WEBSITE

Assignee:
Framework Technologies Corporation

Applicants:
David Halpert, West Windsor, VT
Paul Ames, Lebanon, NH
Mario Ambrosi, Plainfield, NH
Stephen Smith, Lebanon, NH

Abstract of the Disclosure

Disclosed are methods, systems and computer software products for publishing, or displaying, structured data at a website, such that information structures specified by application-specific data generated at a client can be transferred to a server to be automatically expanded into corresponding information structures displayable by a website. Thus, for example, a user may produce a structured document using a program such as Microsoft Project, and wish to have it displayed at a website for display to other members of a work group. According to the present invention, the user executes a web browser, accesses the system's webpage, and drags and drops an icon representing the application-specific structured data onto an Inbox Control displayed on the website. The system of the present invention converts the application-specific data into source-independent data and transfers the data to the server. The server, in accordance with the received source-independent data, either creates or modifies information structures corresponding to the information structures specified by the application-generated structured data for display at the web site.

Background of the Invention

1. Field of the Invention

The present invention relates generally to the website display of structured data. More particularly, the invention relates to a system that can automatically transfer and expand application-specific structured data, that can be obtained from a variety of applications, for display at a website.

2. Background of the Invention

In programming, manufacturing, processing, and other complex development systems, software tools have been developed to efficiently use the resources of the environment to minimize costs and increase efficiency. Design team members often use computerized tools to assist them when developing a complex system. A team member may develop a plan, drawing, schedule or other representation of the system. However, it is difficult to publish the representation to other team members while maintaining the ability to modify it.

In addition, there are many applications for many different types of data. For example, mechanical engineers on a design teams may use three-dimensional CAD systems to develop floor plans that position components of the system on a shop floor or in another appropriate environment to provide easy access and optimize use of available space. Process engineers use process modeling software tools to generate a computerized representation of the system environment to design processes that use raw materials efficiently and eliminate significant bottlenecks. Electrical engineers use software applications to develop electrical plans that ensure, among other things, that the articles in the environment have access to appropriate electrical inputs. Software development teams use project task applications to specify task interrelationships and start and end dates. Technical writers use as technical documentation software to access and modify technical documents having chapters, sections, and subsections, and associated tables-of contents, which must be updated and modified as the document is revised.

Many other data files have embedded information that describes something about the organization and hierarchical decomposition of the design or information of the data. Many kinds of three-dimensional CAD data, such as in files created by CAD tools such as Solidworks or ProEngineer, contain structured data that can be recognized and used by other applications. This type of software is used to organize and display a three-dimensional design into assemblies, sub-assemblies and parts.

Another example of structured data is the case of a project schedule produced by an application like Microsoft Project file. In this case, the structured data that can be recognized includes the tasks and sub-tasks of the schedule.

Structured data can also be recognized in many kinds of drawing programs, such as Visio. Visio allows creation of diagrams such as organizational charts or network diagrams where the structure that is recognized includes pages and objects on the page. In the case of an organizational chart, the structure may be the departments and the employee reporting structure within each department. In the case of a network diagram, it may be the organization and relationships of the various networked PCs, printers and file servers.

Many other types of data, such as that produced by Microsoft PowerPoint outlines, Microsoft Excel spreadsheets, and Bill of Material database reports have data that has categories of information and an implied organizational 'outline' or hierarchy.

While much time is taken to create these types of application-specific structured data and hierarchical data charts and drawings, it is difficult to share the data with the members of a group who are involved with the plans, specifications, schedules or drawings. Frequently, a drawing, chart or schedule will be produced and distributed, and will then need to be updated. Using present systems, it is difficult to control distribution of modifications to the structures and ensure that all who would be affected by the change are timely notified. It is also difficult to maintain original data when making changes to certain data elements.

Systems have been developed that allow software development tools to be used in different environments. Thus, for example, U.S. Patent No. 6,170,081 discloses a system that creates a context object to store intermediate information that is generated while the tool is being used. Information about the environment in which the tool is going to be used is identified and stored in the context object. A native UML model is extracted from a storage device and is converted to an XML file for input to the software development tool. The XML file can then be extracted and converted back to the native UML software model.

U.S. Patent No. 6,003,046 shows a system for on-line browsing of structured documents. The system retrieves a selected page of a structured document, automatically develops context information about the selected page, and then inserts the context information into a webpage. The modified webpage is then sent to a remote location for display. Using the system, a hierarchically-structured document, such as technical documentation with chapters, sections, and subsections, is displayed along with the user's current position in the document using a detailed table-of contents showing the structure of the document. A website stores the structured document, and when a user requests a page of the structured document, the system concatenates the HTML source for that page with a fisheye view of the table-of-contents. Other systems are described in the following U.S. Patents:

6,182,095
6,182,092
6,160,552
6,144,962
6,128,619
6,128,611
6,105,044
6,094,649
6,092,074
6,088,717
6,085,196
6,073,144
6,052,531
5,973,695
5,953,525
5,933,841
5,911,145
5,835,898
5,708,806
5,530,852
5,220,657

However, systems typical of the prior art, such as those described in the above-referenced patents, do not provide a system that publishes, on a website, application-specific, hierarchical data that has been obtained from any of a number of application programs. In addition, none of the prior art systems enables a simple way for a user to modify objects in an original set of application-specific data and have that data appropriately modified and displayed on the website.

Accordingly, a need exists for a software application that can publish hierarchical data that has been obtained from any of a number of applications and have the ability to update that data.

It is therefore an object of the present invention to provide a system that allows a user to publish, at a website, application-specific hierarchical data that can be produced from a variety of applications.

It is a further object of the invention to provide a system allows users to drag and drop any document or file containing structured data onto a Inbox control on a website and have that structure expanded into a matching structure on the website.

It is yet another object of the invention to enable the user to modify objects in the original structure and have the data that is associated with that object on the website appropriately modified.

It is another object of the invention to automatically notify users by e-mail when a data-object and is revised and the revision is republished on the website.

It is an additional object of the invention to allow data modules to interpret new kinds of application-specific data to be easily integrated into the system.

Summary of the Invention

These and other objects are attained by the invention, which comprises methods, systems and computer software products for automatically transferring and expanding application-specific data for display at a website.

According to one aspect of the present invention, the user can execute a web browser, access the server's webpage, and drag and drop an icon representing the application-specific structured data onto a displayed "Inbox Control" area (or use an equivalent designation technique, such as browsing a file list and highlighting a file name). The system of the present invention then passes the designated application-specific data to a Data Selector module that determines: 1) the characteristics of the application-specific data; and 2) the identities of various registered Data Clients to process the data. The Data Selector then passes the application-specific data and the data characteristics to the registered Data Clients. The Data Clients produce bids representing their ability to process the application-specific data, and the Data Selector selects one of the Data Clients to process the data. The selected Data Client converts the application-specific data into source-independent data and transfers the data to the server. The server, in accordance with the received source-independent data, either creates or modifies information structures corresponding to the information structures specified by the application-generated structured data for display at the web site, allowing other users to view the application-specific structured data.

Thus, for example, when a user produces application-specific structured data using a program such as Microsoft Project, in accordance with the invention, the data can be transferred to the system's website for display to multiple users. When the original data is modified, the website is modified accordingly, and notification is sent to the interested users.

Those skilled in the art will appreciate that the methods, systems and software products described herein can be implemented in systems and software other than the specific examples set forth herein, and such examples are provided by way of illustration rather than limitation. In particular, numerous equivalent architectures and configurations of servers, clients, system processing modules, application-specific data and network configurations can be constructed and supported in accordance with the following description.

This specification, including the drawings attached hereto, will next present the invention at various levels of enabling detail, from conceptual to specific examples of implementation.

Brief Description of the Drawings

Fig. 1 shows the major components of the structured data publishing system of the present invention.

Fig. 2A is a flowchart of the processes in the Client computer.

Fig. 2B is a flowchart of the processes in the Server computer.

Fig. 3 is a sample of XML code as would be produced according to the present invention.

Fig. 4 is an example of data produced by the Microsoft Project software application.

Fig. 5 shows the web browser of the present invention where the data produced by Microsoft Project shown in Fig. 4 is dragged and dropped into the Inbox Control of the webpage.

Fig. 6 shows a user-interface display screen, allowing the user to select various publishing options.

Fig. 7 is a user-interface display on which the user indicates that all of the configuration options have been set, and the user is ready to complete the data model construction and web site generation.

Fig. 8 is a partial listing of the ApXML data file that was generated from the Microsoft Project data of Fig. 4.

Fig. 9 shows the new web site content that was built from the project schedule according to the example shown in Figs. 4-8.

Fig. 10 shows a sample drawing produced by the Visio application program.

Fig. 11 is the website screen of the present invention where the user drags and drops the Visio drawing data file of Fig. 10 into the Inbox Control of the webpage.

Fig. 12 is a dialog display screen allowing the user to select various publishing options.

Fig. 13 is a second dialog display screen to indicate completion of entering options.

Fig. 14 shows a partial listing of the ApXML generated from the Visio drawing shown in Fig. 10.

Fig. 15 shows the new web site content that was built from the Visio drawing of Fig. 10.

Detailed Description of the Invention

Structured data publishing according to the present invention allows users to transfer to a web server (using "drag and drop" or an equivalent GUI method) any document or file containing structured data, such that a structure defined by the data can be "expanded" into a matching structure on a corresponding website. Although the following description is directed towards the publishing, or display, on a website, of structured data, the system of the present invention is equally applicable for the web publishing of non-structured data, or non-hierarchical data. Structure is described in generic way, using Extensible Markup Language (XML), and is used on the server to create the structure of the website.

Fig. 1 depicts the interaction of all of the major components involved in the structured data publishing system of the present invention. Client system 100 is connected to server computer 200 over a network connection such as the Internet 300. A user operating an applications program at the client generates application-specific data at the client 100, and would like to have it displayed, or published, by server 200.

The main components on Client 100 are the Inbox Control 110, the Data Selector 120, the Data Clients 130, 140, 150, and the Bundler Control 160. Although the configuration in Fig. 1 shows three Data Clients, any number of Data Clients may be on a particular Client. The main components on Server 200 are the Active Manager 210, Data Processors 220, 230, 240, the ApXML Processor 250, and the Bundler Control 260.

Referring to Figs. 2A and 2B, a flow charts are shown that will be used to describe the structured data publishing system of the present invention in conjunction with Fig. 1.

As shown at 310 in Fig. 2A, after producing application-specific structured data at a client, the user executes a browser program at the server and navigates in the web browser to the system web page from the server that has an "Inbox Control" icon. The "Inbox Control" is an ActiveX control that functions as a "drop" target in a Graphical User Interface image presented by the website to the user at the client. The user "drags and drops" the application-specific data into the Inbox Control 110 of the website. Although in the preferred embodiment, data is imported into the website by dragging and dropping a file representation into the Inbox Control, other methods to import application-specific data onto a website are also possible, such as conventional browsing in a window.

Multiple sets of data, including different types of data from different applications, may be dropped into the "Inbox". The Inbox Control module 110 is the drop target of the website. It receives the dropped data, pre-processes the dropped data, and passes the document or documents to the Data Selector 120. In the preferred embodiment, HTML and Java Script in the page instantiate the Inbox Control and initialize it. Inbox Control 110 also receives the restrictions definitions, which are passed along with the data to the Data Selector.

When an application-specific data file is dragged and dropped into the "Inbox", an appropriate client component, capable of building an XML file for a specific dropped file type, is selected. The resulting XML file is packaged at the client and transferred to the server for processing as a hierarchy doc type. In addition to dragging and dropping an application-specific data file into the "Inbox", the system provides two other mechanisms for importing an XML file to be displayed at the server, a Project Translator and an application primary interface (API). The Project Translator is a server application that imports a new project through a server-side application. The API is a server COM object that imports either a project assembly or a new project. The API is a collection of commands that enables a user to obtain services from an application.

As shown at 320 in Fig. 2A, the Data Selector 120 determines which document handler to use to interpret both the data and the hierarchy of the data. The Data Selector 120 first checks to see which Data Clients are registered. The Data Selector 120 then sends the data, any restrictions, along with descriptives of the data, to each of the registered Data Clients 130, 140, 150 that are on that particular client. Each Data Client examines the dropped data and prepares a bid, indicating its ability to process the data. The Data Client bases its bid value on different criteria, including the file extension, the content of the file, the restrictions that were imposed, and other factors. The Data

Client's bid indicates its ability to handle a given structured data file. The greater the Data Client's ability to handle the file, the higher the Data Client's bid will be.

All of the Data Clients then send their bids to the Data Selector 120. The Data Selector 120 controls the bidding and selects highest bidder, which signifies the best data handler. Restrictions are used in the bidding process to restrict certain types of data. Restrictions passed to the client can affect the bidding and restrict certain types of documents in certain places. Multiple documents can be dropped at the same time and different documents can be handled by different Data Clients.

While Fig. 1 shows three Data Clients 130, 140, 150, there may be any number of Data Clients on a particular client. The Data Clients are extendable data handlers. They bid for data that they recognize and once selected by the Data Selector 120, as shown at 330 on Fig. 2A, one of the Data Clients also processes the data into a source-independent format. Because there may be multiple Data Clients that process different types of data, the structured data publishing system of the present invention is very flexible and easily extendable. Selection of the Data Client processing the dropped document is based on bidding and can be easily extended. Components participating in the selection implement the same interface and can be added and registered with the system at any time. The architecture allows a new 'dataClient' to be easily created to a handle a new file type or the additional interpretation of an existing file type.

The Data Client is a document handler that, in the preferred embodiment, creates a specific, system-defined, version of XML called ApXML. XML is an extensible markup language, and in alternate embodiments other variations of XML could be used. The ApXML file describes the data structure of the file and also creates the supporting files. ApXML defines the structure on the client, as it applies to the meta-design model of the present invention, in a neutral format. The client sends the ApXML file to the server, where the ApXML data is interpreted and the corresponding data structure is created or updated. The ApXML file that is sent to the server is universally usable to create any kind of structure the meta-design of the present invention allows without regard to the original source of the data. ApXML is further described below and in Appendices A and B.

The client-side Data Client interprets, or processes, the application-specific structured data in the file that was dropped in the Inbox Control by interrogating the file in one of two ways. Using the first method of interrogating the file, the Data Client interprets the application-specific data file directly. According to the second method of interrogation, the Data Client opens the original creating application and interrogates it through the application's API to extract information about the file structure, graphic views, and other information. The Data Client also extracts any associated predefined information items associated with the original source data file, such as project element thumbnails, preview pictures or hotspots. It can also extract additional data such as additional project manager information or the like. The Data Client also implements a user interface to get specific information from the user, including options that affect the structure handling or graphical representations. The Data Client uses the information about the document structure to create an ApXML file that describes the data hierarchy and contains all of the related structured data. All of the files are placed in a designated directory structure.

The Data Client, after processing the application-specific data, returns a list of the files and all of the associated information back to the Data Selector 120. The additional information includes the Process Type, the Document Type and the Document Subtype. The Process type determines what Data Processor on the server will be used to process the structured data. The Data Selector 120 passes all the information back to the Inbox Control 110. The Data Client then passes the ApXML formatted data, which is a source-independent format, to Bundler Control 160. (The term "bundling" is used herein to denote a process of packaging files together in a single file or "bundle". Similar processes of combining data files for transfer over network are well-known in the art, and may be used in alternate embodiments.) The Bundler Control bundles all of the data that is necessary for transferring the structured data to the server into one file, as shown at 340 in Fig. 2A. The bundled data includes the ApXML, attached graphic files and information items. The bundled data is then transmitted to the server, as shown at 350 in Fig. 2A.

Referring again to Fig. 1 and to the flow chart in Fig. 2B, the server 200 interprets the ApXML file that is received and creates or updates from the ApXML file and the supporting files the structure of the website itself. The main components of the Server 200 are the Active Manager 210, any number of Data Processors 220, 230 and 240, an ApXML processor 250, and Bundler Control 260. When a data package is received at Server 200, shown at 360 in Fig. 2B, the Bundler Control 260 un-bundles the bundled files that were passed from the client, as shown at 370 in Fig. 2B.

The Active Manager 210 then determines which Data Processor should process the data, shown at 380 in Fig. 2B. The Active Manager 210 restores the content that was created by the Data Client on the client 100. Active Manager 210 then picks a Data Processor to interpret the data. The particular Data Processor is chosen according to the Process Type definition that was passed from the Data Client.

Fig. 1 shows Hierarchy Data Processor 220, Flat Hierarchy Data Processor 230, and Data Processor 240. In most cases of structured data publishing, either the Hierarchy Data Processor 220 or the Flat Hierarchy Data Processor 230 will be used, however other Data Processor modules may be implemented and used in the system. In the illustrated embodiment, the Hierarchy Data Processor creates a thumbnail image representative of the root level of the hierarchy, and processes down from the root level. In contrast, the Flat Hierarchy Data Processor skips the root level and begins processing and displaying at the first hierarchical level after the root level. The illustrated Hierarchy Data Processors thus process the same definition of a given structure, but create different visual representations, to suit user preferences. For example, the user of a Microsoft Project File may wish to have the first chapter of the data exposed when the data is initially displayed. In alternate embodiments, other implementations of Data Processors are possible.

While Fig. 1 shows three Data Processors 220, 230 and 240 on the server, any number of data processors may be in a particular implementation. The Data Processors are Component Object Model (COM) element based on the Microsoft model for defining and encapsulating objects and their interfaces. See www.microsoft.com/com/about.asp (incorporated herein by reference). The Data Processors control how the processed dropped data are applied to the website. Adding a new Data Client and/or a new handled data type does not necessarily require a new data processor to be added. One of the existing, standard processors can be used to interpret new data types. A new data processor is needed only in cases where the data or the hierarchy data need to be applied differently. For example, when some structure other than the standard website hierarchical data is to be created, a new data processor is obtained.

The data is passed to the Data Processor that was selected by the Active Manager 210 , and the selected Data Processor makes the necessary preparations to process the ApXML data . The selected Data Processor calls the ApXML Processor 250 to create the structured part of the website from the object model defined by the ApXML data file and from the other files that were passed from the client. The Data Processor also sends the events that cause the website to be updated and regenerated, such as a signal from the user after completing a modification to the application-specific structured data. The ApXML Processor 250 is also a COM component. As shown at 390 on Fig. 2B, the ApXML Processor processes the ApXML data. The ApXML Processor 250 then determines which information structure corresponds to the received data, shown at 400 in Fig. 2B. The data received may be intended to update an existing data structure on the website, or it may be used create a new website structure. The ApXML Processor 250 creates or updates the data at the website, as shown at 410 in Fig. 2B, according to the instructions that were received from Data Processor 220, 230, 240.

ApXML

ApXML is the mechanism used by the structured data publishing system of the present invention to import any structured data regardless of the data's original source. ApXML is a specific definition of XML tags used to represent the system's complete internal meta-design structure. Appendix A shows the ApXML structure in more detail, as used in the present invention, and Appendix B is a list of the ApXML element tags according to the present invention.

The system uses a meta-design that represents the hierarchy and interdependencies of the structure defined by the "original" data. A significant feature of the invention is that while the ApXML files may not be passed directly to the website, they define a meta-design object model. The meta-design object model, in turn, defines the website structure and the events that modify the website. The meta-design object model itself is persistent, and, among other advantages, enables the provision of back-references and "reconstructability" of the object model. Further information regarding such meta-designs can be found in Framework Technologies Corporation's U.S. Patent No. 5,664,180, the disclosure of which is incorporated herein by reference. Within the meta-design, each aspect can be decomposed into hierarchical project elements and sub-project elements. Each project element may have associated properties and other information items such as files, URLs, or database queries. The meta-design also describes the graphic look for each project element, which could be a thumbnail picture or a hotspot over a background picture. Each element in the meta-design has a corresponding tag in ApXML. For example, the tag 'FACE' is used to define a project element in a specific aspect. Other ApXML tags describe properties of each face of a project element. For example, ApXML uses the property tag 'LOOKGRAPHIC' to set the graphic representation for that face. It uses the property tag 'INFOITEM' to describe any associated information item files with that face. 'INFOITEM' itself has sub-properties that describe the specifics of the information item, such as its name, description, document type, and file path. ApXML is used to describe hierarchical information.

ApXML does not go directly to a website, but instead defines a meta-design object model. The system of the present invention then defines a website and generates the website events based on the meta-design object model. The meta-design object model is persistent, and allows back references to be created and maintained according the invention. ApXML defines the top-level aspects of a project or design by representing them as tabs in the system web sites. The project elements and sub-elements are also defined using ApXML tags. ApXML, according to the present invention, defines project element properties, such as the associated thumbnail picture, and predefined information item files, by associating them with that project element. ApXML defines a reference identifier to allow modifications to be made to an original data structure. When the original data file is modified and then re-published, or re-displayed, with the modifications, the original object is recognized by the system using the reference identifier.

Referring to Fig. 3, a sample ApXML generated from a simple structured data application is shown.

The structured data publishing system of the present invention thus provides a user the ability to import any document or file that contains structured data onto a web site and have that structure automatically expanded into a matching structure on the site itself. The system maintains back references to objects in the original structure so that if the original, application-specific file is modified, the data that is associated with that object on the website is also appropriately modified.

When modifications are made to application-specific data, and then sent to the website to be published, the system recognizes the new place that the modified object exists and properly maps the data to it when the original file is republished with changes. The system detects changes in the structured data and causes the generation of automatic emails to notify particular users of the change.

The structured data publishing system of the present invention will now be further described in the following two examples, where the first example shows application-specific data that was produced using Microsoft Project processed according to the present invention, and the second example shows application-specific data that was produced using Visio. As will be detailed below, a user produces application-specific data, using one of various application systems, and then drags and drops the data into the Inbox Control of the system's webpage for web display at the server.

The system of the present invention determines whether a new website is to be created and published, or an existing website is to be modified and re-published. When an existing website is being modified, the system preserves any information it has throughout the modification. For example, if a user changes a particular PC on a Visio network diagram, the system preserves the association that the PC object has with any comments or information items associated with it even though the Visio diagram was edited and that PC is now in a different place in the diagram. A second example is a user modification of a particular schedule date of a task on a Microsoft project schedule. The system would recognize that the start or end date of the task has changed when the schedule was republished and would notify the subscribers to that task object by automatic e-mail generation and by publishing the revised schedule on the website. These examples will be further described below, although the system is not limited to these applications. It will be readily apparent to those skilled in the art that the structured data publishing system of the present invention can be implemented using any application-specific data.

Example 1: Microsoft Project Data File Published on the ActiveProject Website

According to the present invention, an ActiveProject website is generated by the system using application-specific structured data from any of a variety of application programs. A new

ActiveProject web site structure can be generated directly from Microsoft Project files, and many other types of data files, by extracting structured data from the application data files. A Microsoft Project file decomposes a project into any number of tasks and subtasks. In Microsoft Project, various information can be associated with each task. This information includes Resource names, task start and end dates, task completion status, and other data.

Fig. 4 shows a screen display 450 of data generated using the Microsoft Project program for a sample project called SOFTDEV. On the left is a hierarchical breakdown of the project tasks shown in table form. In this case there are a number of top-level tasks ("Scope", "Analysis/Software Requirements", etc.). Under each task are one or more nested sub-tasks. In this example the "Scope" top-level task has the sub-tasks "Determine project scope", "Secure project sponsorship", etc. These subtasks themselves could be broken down further (in this example they are not). On the right hand side is a graphical depiction (Gantt chart) of the task hierarchy with certain user-defined labeling. Each bar in the Gantt chart is aligned with the comparable task in the table.

In this example, a user desires to create an ActiveProject Web Site, according to the present invention, with a structured publishing node using the data created by Microsoft Project. The structure inherent in the Microsoft Project schedule is extracted by software components in the ActiveProject web site, and is used to generate a matching hierarchy of project elements with associated information items that hold the task information. The steps to perform this operation are listed below. A structured publishing node is a special type of project element that specifies to the underlying software components of the present invention that a new web site structure should be built when the user drops a file that contains recognized structured data.

Fig. 5 shows the ActiveProject website display screen 550 of the present invention with a Structured Publishing Node. The user drag and drops the Microsoft Project file (*.mpp, *.mpt) shown in Fig. 4 on the Inbox icon 560 in the InfoItem frame. As shown in Fig. 5, the Inbox icon 560 is directly above the Add button in the frame on the bottom left of the web site, although in alternate embodiments, it could be located anywhere on the webpage.

Referring again to Figs. 1, 2A and 2B, when a file is dropped on the Inbox 560, a number of steps occur to determine how to process the file. First the Inbox Control 110 receives the

Microsoft Project data file and passes it to the DataSelector 120. The DataSelector 120 determines what DataClients 130, 140, 150, are available on the Client computer 100 and passes the Microsoft Project data file to each of these DataClients 130, 140, 150. Each DataClient 130, 140, 150, examines the Microsoft Project data file and determines if it can process that particular file type (*.mpp, *.mpt). Each DataClient 130, 140, 150 returns an acknowledgment to the DataSelector 120 in the form of a bid. The highest bidding DataClient is the one that is selected to process the file. In this case, where a Microsoft Project file was dropped on the Inbox of a Structured Publishing Node, the highest bidding DataClient will be the MsProjDataClient 140. This DataClient knows how to extract the task hierarchy and task information from a Microsoft Project schedule and to build an ApXML data file to describe this structure to the ActiveProject Server 200. As described above, new DataClients can be developed and added to support any file type.

When the MsProjDataClient 140 is asked to process the Microsoft Project data file by the DataSelector 120, it displays a wizard-like interface so that the user can select various publishing options. The first page of the wizard is shown in Fig. 6. In the page shown in Fig. 6, the user configures the options that will determine the graphical look of the Gantt chart that will be shown in the graphics frame of the web site. After the graphical options are set in the first page, the user can select which additional information to associate with each project element. The additional information is information associated with a task in Microsoft Project. This information will be attached to the project elements in the ActiveProject data model using Information Item objects. These are special types of data model objects for attaching external information to a project element.

Fig. 7 shows a screen on which the user indicates that all of the configuration options have been set, and the user is ready to complete the data model construction and web site generation. When the user clicks on the "Finish" command 710, the MsProjDataClient 450 extracts all of the necessary information from Microsoft Project for the dropped data file. From this information, it generates an ApXML data file that describes the project element hierarchy and all of the associated information items. Each project element corresponds to a task or sub-task in the original Microsoft Project data file. InfoItems are created to hold the additional information associated with each task. Each element in the ApXML that represents a task, sub-task or collection of information (InfoItem) from the original schedule is marked in such a way that an association is created between the ActiveProject data model object and the original schedule object that it represents. Therefore the data model objects "remember" where they came from. ApXML elements are marked with a unique identifier stored in a CustomProp tag. In the case of a task, the unique identifier is the value of the task id in the schedule. For InfoItems it is a unique key derived from the task id. If the same schedule file is modified and dropped on the Inbox 560 again, ActiveProject uses these tags to recognize what has changed and updates the web accordingly. This enables the system to make the appropriate changes to the website after the original publishing operation is modified and subsequently republished

Once the MsProjDataClient 450 has completed the generation of the ApXML file and all associated InfoItem files, it passes the list of files back to the DataSelector 120, which passes it back to the Inbox Control 110. Inbox Control 110 uses Bundler Control 160 to bundle the files into a special type of zip file package. Client 100 then sends the bundled files over the Internet 300 (via the web site) to the ActiveProject server 300 of the present invention. On the server 300, the files are unpacked from the bundle by Bundler Control 260 and passed to ApXMLProcessor 250. The ApXMLProcessor 250 parses the ApXML data file and generates the corresponding information structure using the data model objects specified by the content of the ApXML data file. New web site content, which may be a modification of an existing website or a new website, is generated from the information structure data model objects. The new website content mirrors the original content of the Microsoft Project schedule that was stored in the Microsoft Project data file, and may be published, or displayed, to multiple users.

Referring to Fig. 8, a partial listing of the ApXML data file that was generated from the above Microsoft Project schedule is shown. (The file is shown using Microsoft's XML Notepad viewer). The FaceTarget XML node 810 is a placeholder to represent the project element in the web site where the structure should be built. Underneath the FaceTarget is a list of Face tags. Face tags are used to represent project elements that will be created in the data model hierarchy. There is a one-to-one correspondence between the Face tags in the ApXML data file and the tasks in the original schedule. The Face tags in the ApXML data file are nested in exactly the same way that the tasks are nested in the original schedule. The Face tags for the "Scope" task 820 and its sub-task "Determine project scope" 830 are shown in Fig. 8.

Fig. 9 shows the new web site content that was built from the project schedule according to the example shown in Figs. 4-8. The hierarchy of the project element shown in the outline frame corresponds exactly to the task hierarchy in the original schedule. The graphic in the graphics frame shows the comparable part of the original Gantt chart for the selected project element, or task. Fig. 9 shows, after the ApXML file has been processed, the web site page that is updated and regenerated on the ActiveProject website according to the present invention.

Example 2: Visio Data File Published on the ActiveProject Website

According to the present invention, an ActiveProject web site structure can be generated from other types of data, such as from Visio drawings. In this implementation, the neutrally-formatted data is generated directly from the Visio drawings. A Visio drawing has a natural hierarchical decomposition based on Pages and Shapes. When a drawing is dropped on the ActiveProject Inbox, components in the web site determine what pages are present in the drawing, and what shapes are present in each page. Project elements are constructed based on this information. The steps to perform this operation are listed below.

Fig. 10 shows a drawing called “Basic Network Diagram.vsd.” The drawing shows a typical network layout with various computers connected on a local and wide area network. In this drawing there is only a single page, but each of the computers and network icons represents a unique Visio shape. A user creates the data shown in Fig. 10 and desires to have it published on the ActiveProject website according to the present invention.

As shown in Fig. 11, the website screen is identical to that shown in Fig. 4. However, in this example, the user drags and drops the Visio drawing file (*.vsd) shown in Fig. 10 on the Inbox icon 450 on the web site.

When a file is dropped on the Inbox 450, a number of steps occur to determine how to process the file. First the Inbox Control 110 receives the file and passes it to the DataSelector 120. The DataSelector 120 determines what DataClients 130, 140, 150 are available on the Client computer 100 and passes the file to each of these DataClients 130, 140, 150. Each DataClient 130, 140, 150 examines the file and determines if it can process that particular file type. It returns an acknowledgment to the DataSelector 120 in the form of a bid. The highest bidding DataClient is the one that is selected to process the file. In the case of a Visio drawing dropped on the Inbox of a Structured Publishing Node, the highest bidding DataClient will be the VisioDataClient. The VisioDataClient knows how to extract the Pages and Shapes from a Visio drawing and build an ApXML data file to describe this structure to the ActiveProject server 200.

Referring to Fig. 12, when the VisioDataClient is asked to process the file, it displays a series of dialog screens so that the user can select various publishing options. When processing a single page drawing, like the one in this example, the first dialog would look like that shown in Fig. 12. If the user was processing a multi-page visio drawing, the user would have the option to create web site content from all of the pages or only the first page. In this example there is only one page so the radio button in the dialog is disabled.

Referring to Fig. 13, when the user clicks Continue 600 on the first dialog, a second dialog, as shown in Fig. 13, appears that allows the user to select which shapes to convert to project elements. The shapes to convert are determined by which layer they are on. The user would select one or more layers and all of the shapes on the selected layers would be converted. Typically, Visio drawings are organized into layers. Similar types of objects (shapes) are stored on their own layer. In this example all of the shapes that represent network elements are stored on the Network layer. Other shapes are stored on the background (Off-Layer shapes). When all of the configuration options have been set, the user clicks the OK button to allow the software to complete the data model construction and web site generation.

At this point the VisioDataClient will extract all of the necessary information from Visio for the dropped file. From this information it will generate ApXML data file that describes the project element hierarchy. The top-most project element will represent the drawing itself. The project elements under this will either represent all of the shapes on the layers selected to be converted, or it will represent each page in a multi-page drawing. Each element in the ApXML data file that represents a page or shape from the original drawing is marked in such a way that an association is created between the ActiveProject data model object and the original visio object that it represents. Therefore the data model objects "remember" where they came from. ApXML elements are marked with a unique identifier stored in a CustomProp tag. In the case of a page it is the page number, in the case of a shape it is the shape id. If the same Visio drawing file is modified and dropped on the Inbox again, ActiveProject uses these tags to recognize what has changed and update the web site accordingly. The system thus ensures that all changes made to the web site after the original publishing operation are maintained properly on subsequent republishes.

Once the DataClient has completed the generation of the ApXML file, it passes the original file and the ApXML file back to the DataSelector 120, which passes them back to the Inbox Control 110. The Inbox Control 110, using Bundler Control 160, bundles the files into a special type of zip file package and sends them over the Internet 300 (via the web site) to the ActiveProject Server 200. On Server 200, the files are unpacked from the bundle using Bundler Control 260 and passed to the ApXMLProcessor 250. The ApXMLProcessor 250 parses the ApXMLdata file and generates the information structure data model objects specified by the content of the ApXML. From these information structure data model objects, new web site content is generated that mirrors the original content of the Visio drawing shown in Fig. 10.

Fig. 14 shows a partial listing of the ApXML generated from the Visio drawing shown in Fig. 10. The Face XML 710 node at the top of the page is used to create the project element that represents the entire drawing itself. Face tags are used to represent project elements that will be created in the data model hierarchy. Underneath the top level Face tag is a list of child Face tags. There is a one-to-one correspondence between the Face tags in the ApXML and the shapes on the layers that were selected to be converted to project elements. Face tag 720 shows the face tag for the "Desktop PC" shape.

After the ApXML file is processed, the web site pages that need to be updated are regenerated. Fig. 15 shows the new web site content built from the Visio drawing of Fig. 10. The hierarchy of project elements shown in the outline frame corresponds to the drawing itself, and the shapes on the selected layers. The graphic in the graphics frame shows the Visio drawing with the project element that corresponds to the "Desktop PC" shape selected.